# Cheat Sheet

Programming and Data Science

## Importing packages/modules

Python is known for a wide variety of open source packages for most use cases one can think of. A package or module is basically a file or a collection of files with predefined functions and classes, which you can import into your own code.

```
import package
# then you can call functions and classes from the package
package.function()
```

Alternatively, you can import only specific things from a package:

```
from package import function
# then you can call only this function (but without the package name)
function()
```

You can also import everything from a package:

```
from package import *
# then you can call functions and classes from the package without the package name
function()
otherfunction()
```

Finally you can also give variable names to the things you import:

```
import package as otherName
# then you can call functions and classes from the package
otherName.function()

from package import function as somethingelse
somethingelse()
```

There are many useful packages already in the python standard library (`https://docs.python.org/3/library/index.html`). For example

- math: for mathematical functions
- statistics: for statistics...
- os: for calling your operating system
- random: for generating random numbers

Other packages need to be installed using a package manager. The default package manager which (should hopefully) come with the basic python installation is pip (package installer for Python). You can install new packages by calling pip in the command line or in the terminal in VSCode.

```
pip install packagename
```

Useful packages for this lecture are:

- numpy: for numerical operations on vectors and matrices
- matplotlib: for plotting
- scikit-learn: for machine learning and data science

(Other suggested packages are scipy and pandas))

## Indexing

To access elements of data structures such as lists or tuples, you can use indices, integer numbers starting from 0. By calling the for example a list and putting square brackets with an index behind, you access the corresponding element. Putting a minus in front of the index will start counting from the back.

```
numbers = [1,2,3]
print(numbers[0]) # 1
print(numbers[1]) # 2
print(numbers[-1]) # 3
```

To access ranges of elements, the syntax is to use colons as in the following example:

```
numbers = [1,2,3,4,5,6,7,8]
start = 0
stop = 4
step = 1
print(numbers[start:stop:step]) # [1, 2, 3, 4]

print(numbers[3:7]) # [4, 5, 6, 7]
```

Using only one colon, the step is omitted and set to 1.

## String methods

There are a lot of operations which you can perform on strings. For example you can access different characters via indices similar to lists. Other useful functions are replace and split.

```
string = 'abc def'

print(string[0]) # a

print(string.replace('a', 'b')) # bbc def

print(string.split(' ')) # ['abc','def']
```

see `https://docs.python.org/3/library/stdtypes.html#string-methods`

## Dictionaries

Dictionaries are a kind of data structure consisting of key-value pairs. To access a given value of the dictionary you don't use an index, but the corresponding key. Neither key nor value need to have a specific type, but for our purposes most often keys are strings.

```
# dictionary = {'key':'value'}
alphabet = {'a' : 1, 'b' : 2, 'c' : 3}
print(alphabet{'a'}) # 1
```

## Indentation

Python code is indentation sensitive! This means that certain code blocks need to be properly indented in order to work. Such code block might be loops, contents of a function or if-else statements.

## Useful builtin functions

- abs(x) : return absolute value of a number x
- dir(x) : lists all attributes and functions of a class instance x
- locals() : returns a dictionary of all local variables
- globals() : returns a dictionary of all global variables

## Defining a new function

If you need to execute a certain code block multiple times with varying arguments, you can define a function to avoid redundant code.

```
def functionName(argument1, argument2):
    result = 5 * argument1 + 3 - argument2
    return result

test = functionName(3,5)
```

The return statements defines the output of the function. The contents of the function definition need to be indented. You can call the function by typing the function name with all relevant arguments (inputs) in round brackets.

## If-Else Statements

```
x = 10
condition = x > 4
if condition:
    print('yes')
else:
    print('no')
```

If your code is supposed to distinguish between different situations you can use if statements to check if certain conditions are fulfilled. A condition is a boolean. If the condition is true, everything after the if statement is executed, if not the code jumps to the else statement.

## For Loops

If you want to do the same task repeatedly, this can be done in a loop. A for loop executes the code in the indented loop block for each element in the datastructure you iterate over.

```
numbers = range(0,10,1) # outside the loop
for i in numbers:
    print(i)                # inside the loop
```

range(start, stop, step) is a useful builtin function which generates ranges of numbers over which you can iterate. In this example we iterate over the numbers from 0 to 9 and print each of them in the loop. In each iteration of the loop, the variable i corresponds to another value.

## While Loops

Another loop variant is the while loop. This loop will execute the indented code as long as a condition is fulfilled (boolean==True).

```
i = 0 # outside the loop
while i < 10:
    print(i) # inside the loop
    i = i + 1 # inside the loop
```

This will print the numbers from 0 to 9 and end as soon as the condition is no longer fulfilled. You can also combine loops with if statements. In that case special builtin python functions might become relevant.

```
i = 0
while True: # this would run forever, because True is always True
    print(i)
    if i < 10:
        continue # this will continue the loop
    else:
        break         # this will end the loop
    i = i + 1
```

The result should be again the numbers from 0 to 9.

## Plotting

Make a line plot x vs y

```
import matplotlib.pyplot as plt
plt.plot(x,y)
plt.title('a line plot')
plt.show()
```

Make a scatter plot x vs y

```
import matplotlib.pyplot as plt
plt.scatter(x,y)
plt.title('a scatter plot')
plt.show()
```

here x and y are usually lists or tuples of numbers (int or float). There are a lot of options to customize the look of your plot, like colors, line styles, marker size and shape, etc.
see `https://matplotlib.org/3.5.3/api/_as_gen/matplotlib.pyplot.html`

## Math

For more complicated math operations there is a package called 'math' from which you can import various functions

```
from math import exp
result = exp(2) # this means e^2 so "e to the power of two"
```

other useful functions might be:
square root = sqrt, sine = sin, cosine = cos
see `https://docs.python.org/3/library/math.html`